

## INTRODUCTION OF JAVA

### ● What is JAVA?

Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language

Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java

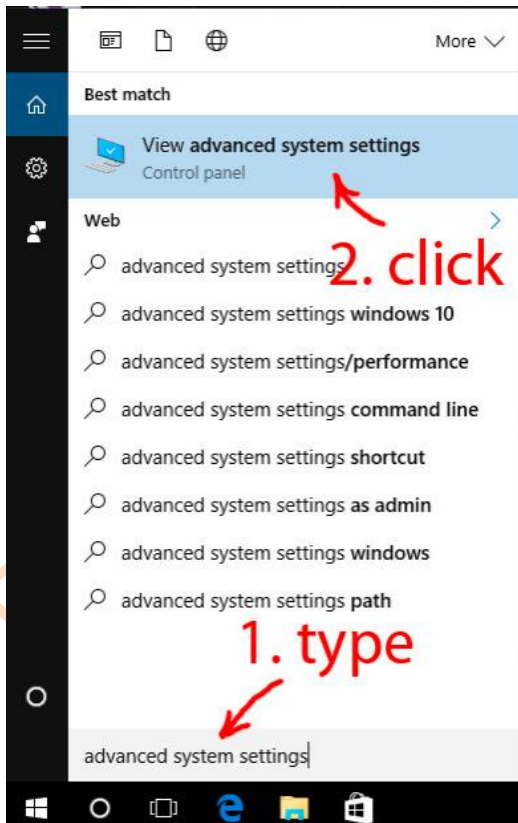
**Platform:** Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform

### ● Setting up the Environment

The path is required to be set for using tools such as javac, java, etc

For setting the permanent path of JDK, you need to follow these steps:

**Step1:** In the search field type in – advanced system settings

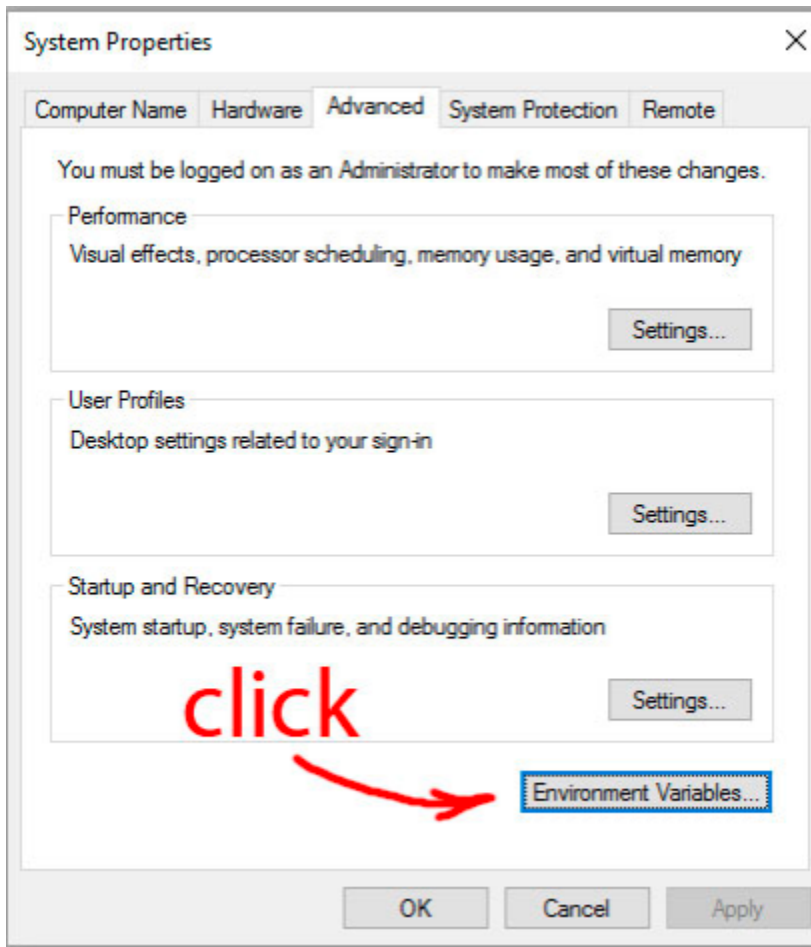


**Step2:** Set JAVA\_HOME Environment variable

## CORE JAVA PROGRAMMING

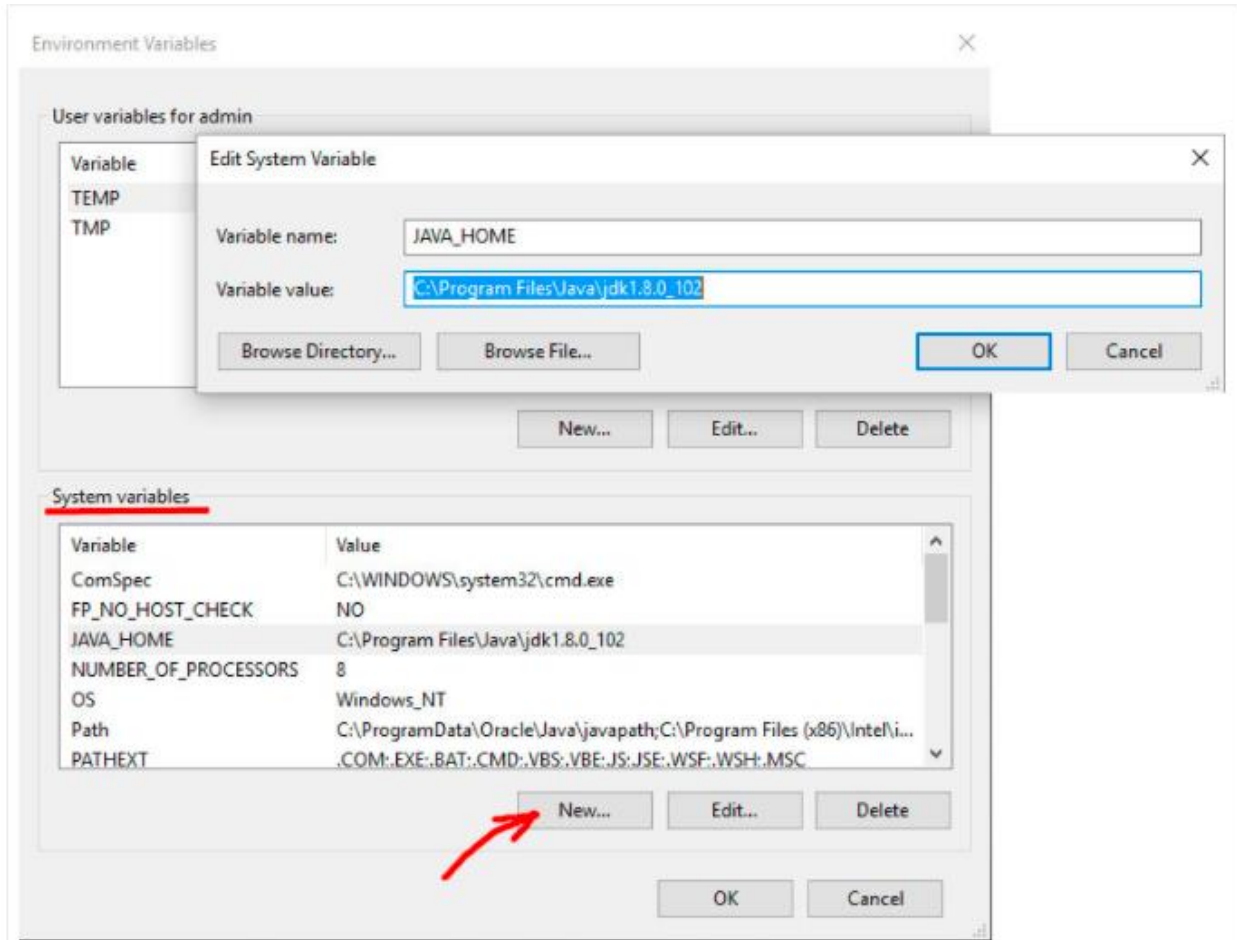
---

In “System Properties window” click “Environment Variables...”



Under “System variables” click the “New...” button and enter JAVA\_HOME as “Variable name” and the path to your Java JDK directory under “Variable value”

## CORE JAVA PROGRAMMING

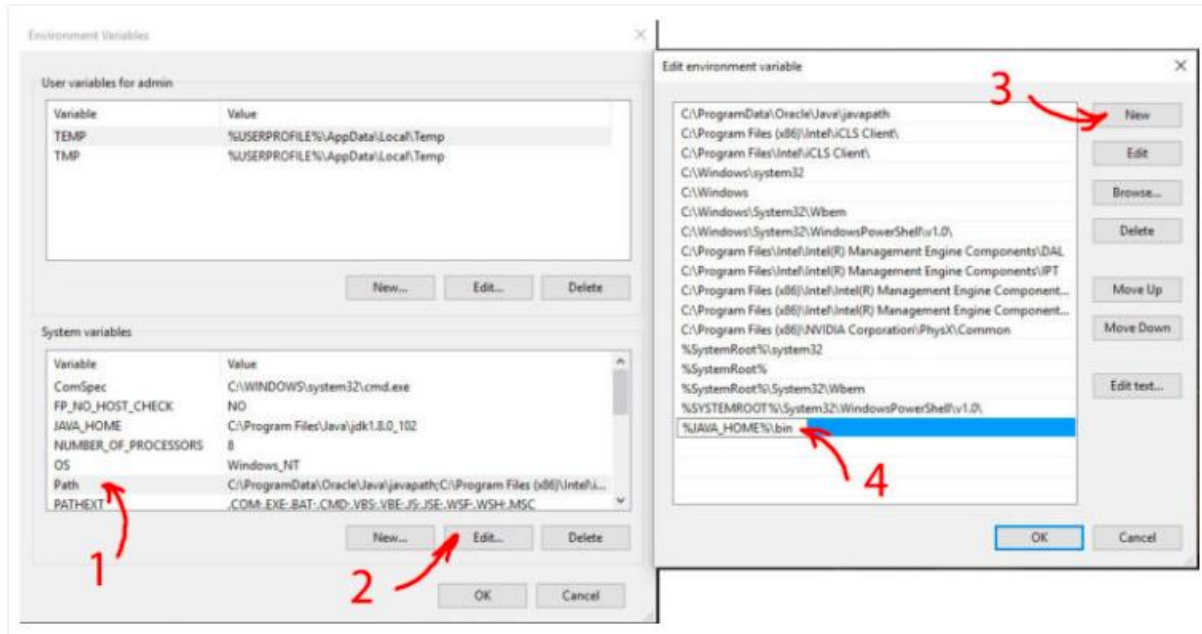


Add JAVA\_HOME as system variable

### Step3: Update System PATH

- In "Environment Variables" window under "System variables" select Path
- Click on "Edit..."
- In "Edit environment variable" window click "New"
- Type in %JAVA\_HOME%\bin

# CORE JAVA PROGRAMMING



## Step4: Test your Configuration

Open a new command prompt and type in:

```
echo %JAVA_HOME%
```

this will print out the directory JAVA\_HOME points to or empty line if the environment variable is not set correctly

Now type in:

```
javac -version
```

this will print out the version of the java compiler if the Path variable is set correctly or "javac is not recognized as an internal or external command..." otherwise.

```
Command Prompt
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\admin>echo %JAVA_HOME%
C:\Program Files\Java\jdk1.8.0_102

C:\Users\admin>javac -version
javac 1.8.0_102

C:\Users\admin>
```

## • A First JAVA Program

Lets Create a Hello JAVA Program

Created By –Mr. Gauri Shankar Rai

# CORE JAVA PROGRAMMING

---

## Java "Hello, World!" Program

```
// Your First Program

class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

## Output

```
Hello, World!
```

## How Java "Hello, World!" program works?

### 1. // Your First Program

In Java, any line starting with // is a comment. Comments are intended for users reading the code to understand the intent and functionality of the program. It is completely ignored by the Java compiler

Java Compiler: an application that translates Java program to Java bytecode that computer can execute

### 2. class HelloWorld { ... }

In Java, every application begins with a class definition. In the program HelloWorld is the name of the class. For now, just remember that every Java application has a class definition, and the name of the class should match the filename in Java

### 3. public static void main(String[] args) { ... }

This is the main method. Every application in Java must contain the main method. The Java compiler starts executing the code from the main method.

We will learn the meaning of public, static, void, and how methods work? in later chapters. For now, just remember that the main function is the entry point of your Java application, and it's mandatory in a Java program.

### 4. System.out.println("Hello, World!");

## CORE JAVA PROGRAMMING

---

The code above is a print statement. It prints the text Hello, World! to standard output (your screen). The text inside the quotation marks is called String in Java.

Notice the print statement is inside the main function, which is inside the class definition.

### Things to Remember:

- Every valid Java Application must have a class definition that matches the filename (class name and file name should be same).
- The main method must be inside the class definition.
- The compiler executes the codes starting from the main function.

### ● Java Naming Conventions

Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc.

But, it is not forced to follow. So, it is known as convention not rule. These conventions are suggested by several Java communities such as Sun Microsystems and Netscape.

All the classes, interfaces, packages, methods and fields of Java programming language are given according to the Java naming convention. If you fail to follow these conventions, it may generate confusion or erroneous code.

### Advantage of Naming Convention in Java

By using standard Java naming conventions, you make your code easier to read for yourself and other programmers. Readability of Java program is very important. It indicates that less time is spent to figure out what the code does.

### Naming Conventions of Different Identifiers

Identifiers Type	Naming Rules	Examples
------------------	--------------	----------

## CORE JAVA PROGRAMMING

Class	It should start with the uppercase letter. It should be a noun such as Color, Button, System, Thread, etc. Use appropriate words, instead of acronyms.	public class <b>Employee</b> { //code snippet }
Interface	It should start with the uppercase letter. It should be an adjective such as Runnable, Remote, ActionListener. Use appropriate words, instead of acronyms.	interface <b>Printable</b> { //code snippet }
Method	It should start with lowercase letter. It should be a verb such as main(), print(), println(). If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().	class Employee { // method void <b>draw()</b> { //code snippet } }
Variable	It should start with a lowercase letter such as id, name. It should not start with the special characters like & (ampersand), \$ (dollar), _ (underscore). If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName. Avoid using one-character variables such as x, y, z.	class Employee { // variable int <b>id</b> ; //code snippet }
Package	It should be a lowercase letter such as java, lang. If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.	//package package <b>com.javatpoint</b> ; class Employee { //code snippet }
Constant	It should be in uppercase letters such as RED, YELLOW. If the name contains multiple words, it should be separated by an underscore(_) such as MAX_PRIORITY. It may contain digits but not as the first letter.	class Employee { //constant static final int <b>MIN_AGE</b> = 18; //code snippet }

### CamelCase in Java naming conventions

Java follows camel-case syntax for naming the class, interface, method, and variable.

If the name is combined with two words, the second word will start with uppercase letter always such as actionPerformed(), firstName, ActionEvent, ActionListener, etc.

### ● JVM (Java Virtual Machine) Architecture

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

#### What is JVM?

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

#### What it does?

The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set

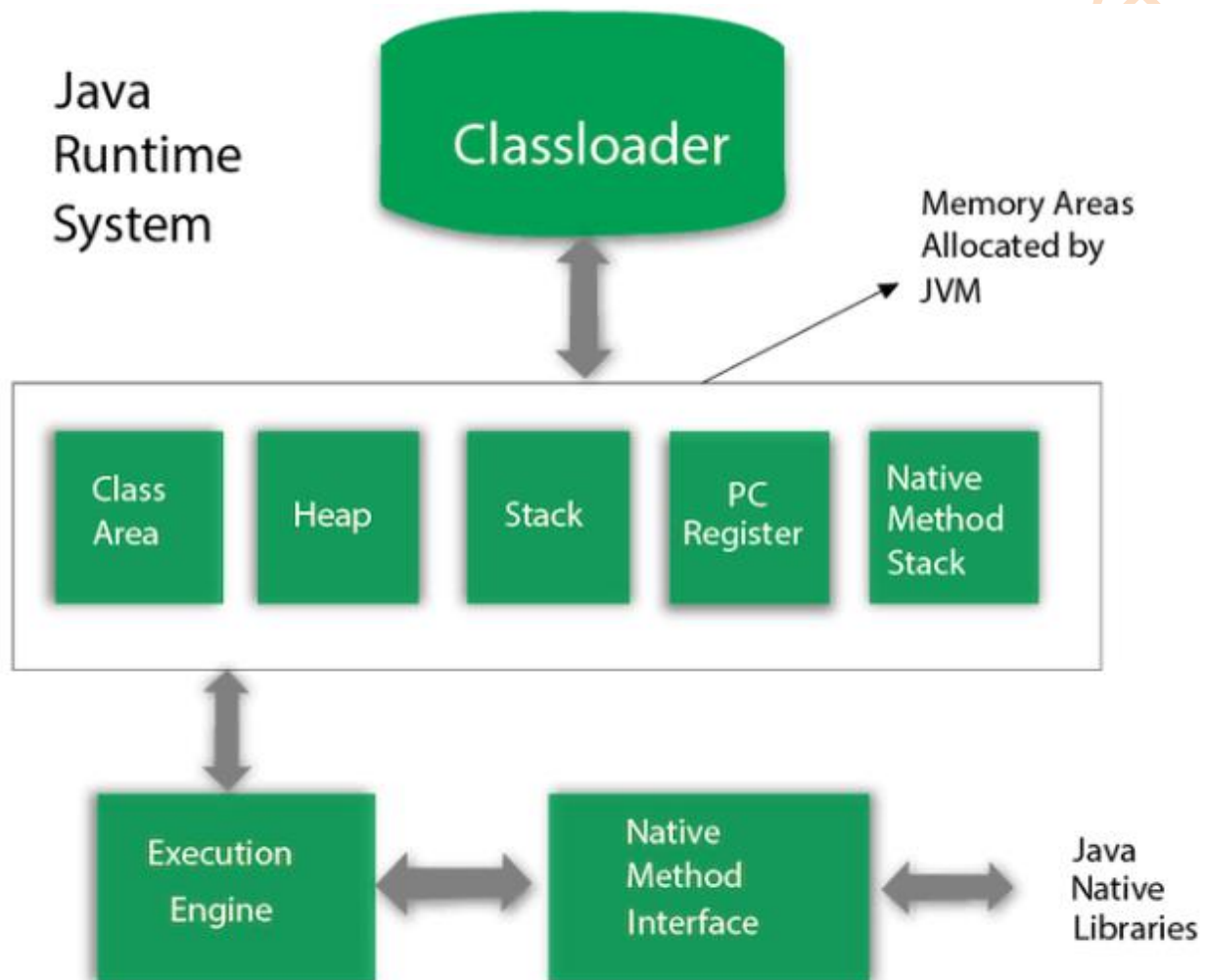


## CORE JAVA PROGRAMMING

- Garbage-collected heap
- Fatal error reporting etc.

### JVM Architecture

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc



#### 1. Classloader

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

# CORE JAVA PROGRAMMING

---

1. **Bootstrap ClassLoader:** This is the first classloader which is the super class of Extension classloader. It loads the rt.jar file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.
2. **Extension ClassLoader:** This is the child classloader of Bootstrap and parent classloader of System classloader. It loads the jar files located inside \$JAVA\_HOME/jre/lib/ext directory.
3. **System/Application ClassLoader:** This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

//Let's see an example to print the classloader name

```
public class ClassLoaderExample
{
    public static void main(String[] args)
    {
        //Let's print the classloader name of current class.
        //Application/System classloader will load this class

        Class c=ClassLoaderExample.class;
        System.out.println(c.getClassLoader());

        //If we print the classloader name of String, it will print null because it //is an in-built class
        //which is found in rt.jar, so it is loaded by //Bootstrap classloader

        System.out.println(String.class.getClassLoader());
    }
}
```

**Output:** null

These are the internal classloaders provided by Java. If you want to create your own classloader, you need to extend the ClassLoader class

## 2. Class(Method) Area

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

## 3. Heap:

It is the runtime data area in which objects are allocated.

## 4. Stack

## CORE JAVA PROGRAMMING

---

Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

### 5. Program Counter Register

PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

### 6. Native Method Stack

It contains all the native methods used in the application

### 7. Execution Engine:

It Contains,

#### 1. A virtual processor

#### 2. Interpreter: Read bytecode stream then execute the instructions.

#### 3. Just-In-Time(JIT) compiler: It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

### 8. Java Native Interface

Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.

### ● Difference between JDK, JRE and JVM

Sr. No.	Key	JDK	JRE	JVM
1	Definition	JDK (Java Development Kit) is	JRE (Java Runtime Environment) is the	JVM (Java Virtual Machine) is an abstract machine that

## CORE JAVA PROGRAMMING

Sr. No.	Key	JDK	JRE	JVM
		a software development kit to develop applications in Java. In addition to JRE, JDK also contains number of development tools (compilers, JavaDoc, Java Debugger etc.).	implementation of JVM and is defined as a software package that provides Java class libraries, along with Java Virtual Machine (JVM), and other components to run applications written in Java programming.	is platform-dependent and has three notions as a specification, a document that describes requirement of JVM implementation, implementation, a computer program that meets JVM requirements, and instance, an implementation that executes Java byte code provides a runtime environment for executing Java byte code.
2	Prime functionality	JDK is primarily used for code execution and has prime functionality of development.	On other hand JRE is majorly responsible for creating environment for code execution.	JVM on other hand specifies all the implementations and responsible to provide these implementations to JRE.
3	Platform Independence	JDK is platform dependent i.e for different platforms different JDK required.	Like of JDK JRE is also platform dependent.	JVM is platform independent.
4	Tools	As JDK is responsible for prime development so it contains tools for developing, debugging and monitoring java application.	On other hand JRE does not contain tools such as compiler or debugger etc. Rather it contains class libraries and other supporting files that JVM requires to run the program.	JVM does not include software development tools.
5	Implementation	JDK = Java Runtime Environment (JRE) + Development tools	JRE = Java Virtual Machine (JVM) + Libraries to run the application	JVM = Only Runtime environment for executing the Java byte code.

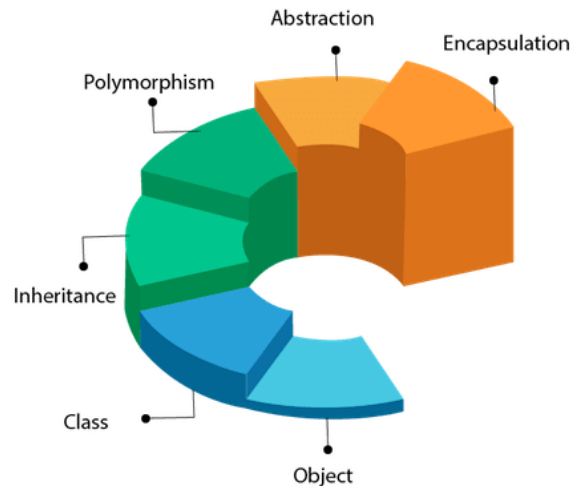
### **OOPs (OBJECT – ORIENTED PROGRAMMING SYSTEM) CONCEPT**

- **Introduction to OOP Concept**

# CORE JAVA PROGRAMMING

**Object** means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation



Apart from these concepts, there are some other terms which are used in Object-Oriented design:

- Coupling
- Cohesion
- Association
- Aggregation
- Composition

## ○ Object

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

**Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.



## ○ Class

*Collection of objects* is called class. It is a logical entity.

# CORE JAVA PROGRAMMING

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

## ○ Inheritance

*When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.*

## ○ Polymorphism

*If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.*

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.



## ○ Abstraction

*Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.*

In Java, we use abstract class and interface to achieve abstraction.

## ○ Encapsulation

*Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.*

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.



Capsule

## ○ Coupling

## CORE JAVA PROGRAMMING

---

Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other. If a class has the details information of another class, there is strong coupling. In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field. You can use interfaces for the weaker coupling because there is no concrete implementation.

### ○ Cohesion

Cohesion refers to the level of a component which performs a single well-defined task. A single well-defined task is done by a highly cohesive method. The weakly cohesive method will split the task into separate parts. The java.io package is a highly cohesive package because it has I/O related classes and interface. However, the java.util package is a weakly cohesive package because it has unrelated classes and interfaces.

### ○ Association

Association represents the relationship between the objects. Here, one object can be associated with one object or many objects. There can be four types of association between the objects:

- One to One
- One to Many
- Many to One, and
- Many to Many

Let's understand the relationship with real-time examples. For example, One country can have one prime minister (one to one), and a prime minister can have many ministers (one to many). Also, many MP's can have one prime minister (many to one), and many ministers can have many departments (many to many).

Association can be unidirectional or bidirectional.

### ○ Aggregation

Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state. It represents the weak relationship between objects. It is also termed as a **has-a** relationship in Java. Like, inheritance represents the is-a relationship. It is another way to reuse objects.

### ○ Composition

The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state. There is a strong relationship between the containing object and the dependent object. It is the state where containing objects do not have an independent existence. If you delete the parent object, all the child objects will be deleted automatically.

### ○ Advantage of OOPs over Procedure-oriented programming language



## CORE JAVA PROGRAMMING

1. OOPs makes development and maintenance easier, whereas, in a procedure-oriented programming language, it is not easy to manage if code grows as project size increases.
2. OOPs provides data hiding, whereas, in a procedure-oriented programming language, global data can be accessed from anywhere.
3. OOPs provides the ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

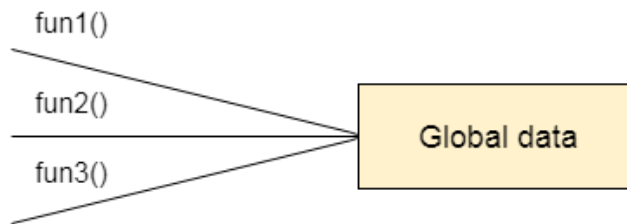


Figure: Data Representation in Procedure-Oriented Programming

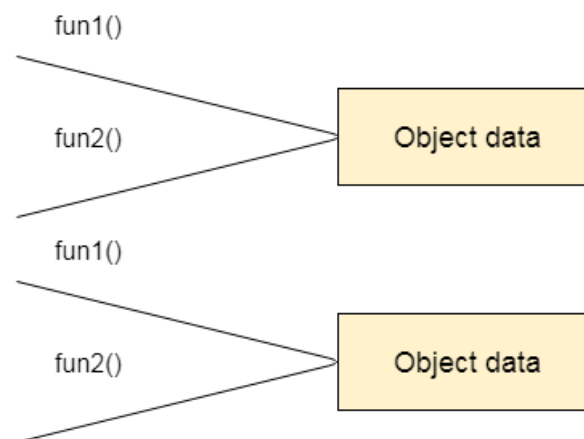


Figure: Data Representation in Object-Oriented Programming

**What is the difference between an object-oriented programming language and object-based programming language?**

Object-based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object-based programming languages.

## DATA TYPES, VARIABLES AND ARRAYS

## ● Java Identifiers

Identifiers in Java are symbolic names used for identification. They can be a class name, variable name, method name, package name, constant name, and more. However, In Java There are some reserved words that cannot be used as an identifier.

For every identifier there are some conventions that should be used before declaring them. Let's understand it with a simple Java program:

```
public class demo {  
    public static void main(String[] args) {  
        System.out.println("Hello Netzwerk Academy");  
    }  
}
```

From the above example, we have the following Java identifiers:

1. HelloJava (Class name)
2. main (main method)
3. String (Predefined Class name)
4. args (String variables)
5. System (Predefined class)
6. out (Variable name)
7. println (method)

### Rules for Identifiers in Java

There are some rules and conventions for declaring the identifiers in Java. If the identifiers are not properly declared, we may get a compile-time error. Following are some rules and conventions for declaring identifiers:

- A valid identifier must have characters [A-Z] or [a-z] or numbers [0-9], and underscore(\_) or a dollar sign (\$). for example, @netzwerk is not a valid identifier because it contains a special character which is @.
- There should not be any space in an identifier. For example, netz werk is an invalid identifier.
- An identifier should not contain a number at the starting. For example, 123netzwerk is an invalid identifier.
- An identifier should be of length 4-15 letters only. However, there is no limit on its length. But, it is good to follow the standard conventions.
- We can't use the Java reserved keywords as an identifier such as int, float, double, char, etc. For example, int double is an invalid identifier in Java.
- An identifier should not be any query language keywords such as SELECT, FROM, COUNT, DELETE, etc.

### Java Reserved Keywords

# CORE JAVA PROGRAMMING

Java reserved keywords are predefined words, which are reserved for any functionality or meaning. We cannot use these keywords as our identifier names, such as class name or method name. These keywords are used by the syntax of Java for some functionality. If we use a reserved word as our variable name, it will throw an error.

In Java, every reserved word has a unique meaning and functionality.

Consider the below syntax:

```
double marks;
```

in the above statement, double is a reserved word while marks is a valid identifier.

**Below is the list of reserved keywords in Java:**

abstract	continue	for	protected	transient
Assert	Default	Goto	public	Try
Boolean	Do	If	Static	throws
break	double	implements	strictfp	Package
byte	else	import	super	Private
case	enum	Interface	Short	switch
Catch	Extends	instanceof	return	void
Char	Final	Int	synchronized	volatile
class	finally	long	throw	Date
const	float	Native	This	while

Although the const and goto are not part of the Java language; But, they are also considered keywords.

## Example of Valid and Invalid Identifiers

Following are some examples of **valid** identifiers in Java:

- TestVariable
- testvariable
- a
- i
- Test\_Variable
- \_testvariable
- \$testvariable
- sum\_of\_array

# CORE JAVA PROGRAMMING

---

- TESTVARIABLE
- test123

Below are some examples of **invalid** identifiers:

- Test Variable ( We cannot include a space in an identifier)
- 123test ( The identifier should not begin with numbers)
- test+variable ( The plus (+) symbol cannot be used)
- test-variable ( Hyphen symbol is not allowed)
- test\_&\_variable ( ampersand symbol is not allowed)
- Test'variable (we cannot use an apostrophe symbol in an identifier)

We should follow some naming convention while declaring an identifier. However, these conventions are not forced to follow by the Java programming language. That's why it is called conventions, not rules. But it is good to follow them. These are some industry standards and recommended by Java communities such as Oracle and Netscape.

If we do not follow these conventions, it may generate confusion or erroneous code.

## ● Data Types

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

### 1. Java Primitive Data Types:

In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

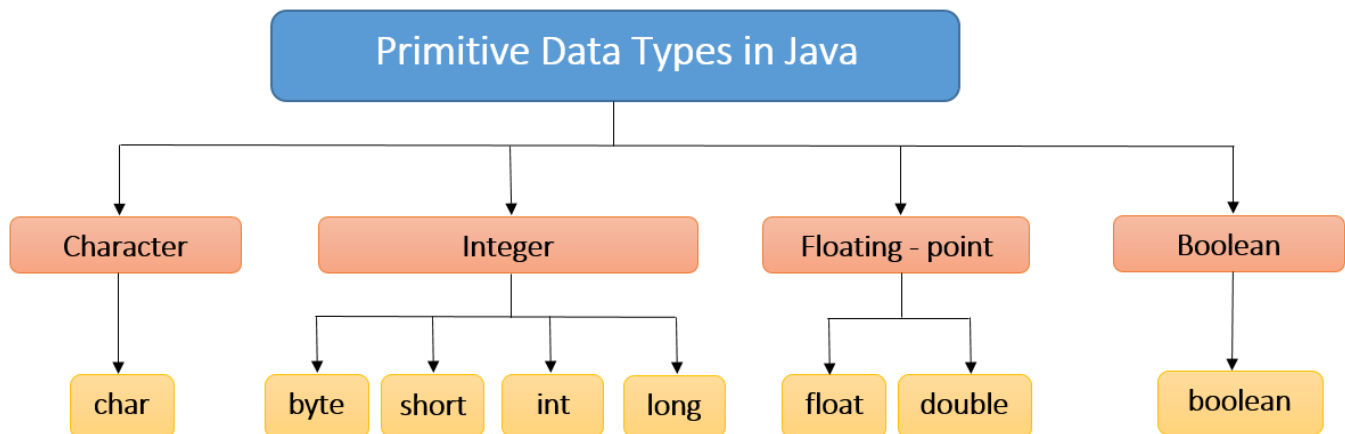
*Java is a statically-typed programming language. It means, all variables must be declared before its use. That is why we need to declare variable's type and name.*

**There are 8 types of primitive data types:**

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type

# CORE JAVA PROGRAMMING

- float data type
- double data type



## Boolean Data Type

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example:

```
Boolean one = false
```

## Byte Data Type

The byte data type is an example of primitive data type. It is an 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example:

```
byte a = 127, byte b = -128
```

## Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

## CORE JAVA PROGRAMMING

---

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example:

```
short s = 10000, short r = -5000
```

### Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between -2,147,483,648 ( $-2^{31}$ ) to 2,147,483,647 ( $2^{31}-1$ ) (inclusive). Its minimum value is -2,147,483,648 and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example:

```
int a = 100000, int b = -200000
```

### Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808 ( $-2^{63}$ ) to 9,223,372,036,854,775,807 ( $2^{63}-1$ ) (inclusive). Its minimum value is -9,223,372,036,854,775,808 and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example:

```
long a = 100000L, long b = -200000L
```

### Float Data Type

The float data type is a single-precision 32-bit IEEE 754 floating point. Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0f.

Example:

```
float f1 = 123.5f
```

### Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example:

```
double d1 = 14.7
```

# CORE JAVA PROGRAMMING

## Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive). The char data type is used to store characters.

Example:

```
char letterN = 'N'
```

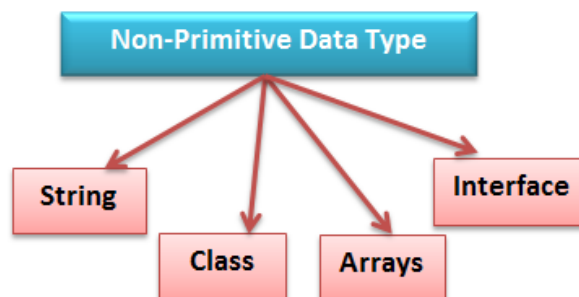
## Why char uses 2 byte in java and what is \u0000 ?

It is because java uses Unicode system not ASCII code system. The \u0000 is the lowest range of Unicode system.

Data Type	Default Value	Default Size	Value Range	Example
boolean	false	1 bit (which is a special type for representing true/false values)	true/false	boolean b=true;
char	'\u0000'	2 byte (16 bit unsigned unicode character)	0 to 65,535	char c='a';
byte	0	1 byte (8 bit Integer data type)	-128 to 127	byte b=10;
short	0	2 byte (16 bit Integer data type)	-32768 to 32767	short s=11;
int	0	4 byte (32 bit Integer data type)	-2147483648 to 2147483647.	int i=10;
long	0L	8 byte (64 bit Integer data type)	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	long l=100012;
float	0.0f	4 byte (32 bit float data type)	1.40129846432481707e-45 to 3.40282346638528860e+38 (positive or negative).	float f=10.3f;
double	0.0d	8 byte (64 bit float data type)	4.94065645841246544e-324d to 1.79769313486231570e+308d (positive or negative)	double d=11.123;

## 2. Non-Primitive Datatypes

Non-Primitive data types refer to objects and hence they are called reference types. Examples of non-primitive types include Strings, Arrays, Classes, Interface, etc.



## String Data Type

String is a sequence of characters. But in Java, a string is an object that represents a sequence of characters. The *java.lang.String* class is used to create a string object.

## Class Data Type

A class in Java is a blueprint which includes all your data. A class contains fields (variables) and methods to describe the behavior of an object.

## Arrays Data Type

Arrays in Java are homogeneous data structures implemented in Java as objects. Arrays store one or more values of a specific data type and provide indexed access to store the same. A specific element in an array is accessed by its index.

## Interface Data Type

Like a class, an interface can have methods and variables, but the methods declared in interface are by default abstract (only method signature, no body).

## Difference between primitive and non-primitive data types

- Primitive types are predefined in Java. Non-primitive types are created by the programmer and is not defined by Java.
- Non Primitive types can be used to call methods to perform certain operations, while primitive types cannot.
- A primitive type always has a value, whereas non-primitive types can be null.
- A primitive type starts with a lowercase letter, while non-primitive types start with an uppercase letter.
- The size of a primitive type depends on the data type, while non-primitive types have all the same size.



- **How to define our own Data type in Java (enum)**

### **enum in Java**

Enumerations serve the purpose of representing a group of named constants in a programming language.

The Enum in Java is a data type which contains a fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY) , directions (NORTH, SOUTH, EAST, and WEST), season (SPRING, SUMMER, WINTER, and AUTUMN or FALL), colors (RED, YELLOW, BLUE, GREEN, WHITE, and BLACK) etc. According to the Java naming conventions, we should have all constants in capital letters. So, we have enum constants in capital letters.

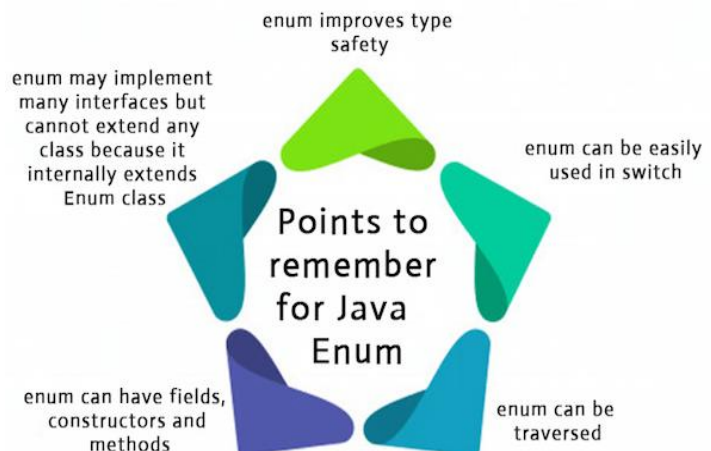
Java Enums can be thought of as classes which have a fixed set of constants (a variable that does not change). The Java enum constants are static and final implicitly. It is available since JDK 1.5.

Enums are used to create our own data type like classes. The enum data type (also known as Enumerated Data Type) is used to define an enum in Java. Unlike C/C++, enum in Java is more powerful. Here, we can define an enum either inside the class or outside the class but not inside a Method

Java Enum internally inherits the Enum class, so it cannot inherit any other class, but it can implement many interfaces. We can have fields, constructors, methods, and main methods in Java enum.

### **Points to remember for Java enum**

- Enum improves type safety
- Enum can be easily used in switch
- Enum can be traversed
- Enum can have fields, constructors and methods
- Enum may implement many interfaces but cannot extend any class because it internally extends Enum class

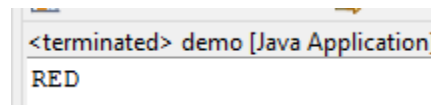


### Simple Example of Java Enum

- A simple enum example where enum is declared outside any class (Note enum keyword instead of class keyword)

```
enum Color {  
    RED,  
    GREEN,  
    BLUE;  
}  
  
public class demo {  
    // Driver method  
    public static void main(String[] args)  
    {  
        Color c1 = Color.RED;  
        System.out.println(c1);  
    }  
}
```

Output:

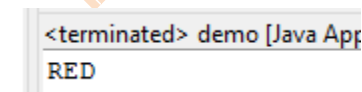


<terminated> demo [Java Application]  
RED

- enum declaration inside a class.

```
public class demo {  
    enum Color {  
        RED,  
        GREEN,  
        BLUE;  
    }  
  
    // Driver method  
    public static void main(String[] args)  
    {  
        Color c1 = Color.RED;  
        System.out.println(c1);  
    }  
}
```

Output:



<terminated> demo [Java App]  
RED

## CORE JAVA PROGRAMMING

- A Java program to demonstrate working on enum in switch case (Filename demo.java)

```
//An Enum class
enum Day {
    SUNDAY,
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY;
}

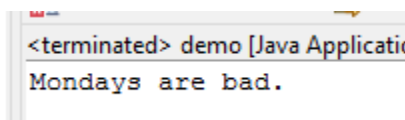
//Driver class that contains an object of "day" and
//main().
public class demo {
    Day day;

    // Constructor
    public demo(Day day) { this.day = day; }

    // Prints a line about Day using switch
    public void dayIsLike()
    {
        switch (day) {
            case MONDAY:
                System.out.println("Mondays are bad.");
                break;
            case FRIDAY:
                System.out.println("Fridays are better.");
                break;
            case SATURDAY:
            case SUNDAY:
                System.out.println("Weekends are best.");
                break;
            default:
                System.out.println("Midweek days are so-so.");
                break;
        }
    }

    // Driver method
    public static void main(String[] args)
    {
        String str = "MONDAY";
        demo t1 = new demo(Day.valueOf(str));
        t1.dayIsLike();
    }
}
```

Output:



The screenshot shows a Java application window titled "<terminated> demo [Java Application]". The output text "Mondays are bad." is displayed in the console area.

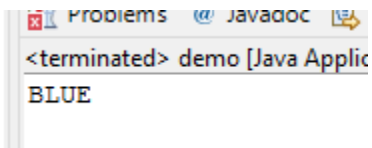
## CORE JAVA PROGRAMMING

---

- A Java program to demonstrate that we can have main() inside enum class.

```
enum demo {  
    RED,  
    GREEN,  
    BLUE;  
  
    // Driver method  
    public static void main(String[] args)  
    {  
        demo c1 = demo.BLUE;  
        System.out.println(c1);  
    }  
}
```

Output:



### enum and Inheritance:

- All enums implicitly extend **java.lang.Enum** class. As a class can only extend one parent in Java, so an enum cannot extend anything else.
- **toString()** method is overridden in **java.lang.Enum** class, which returns enum constant name.
- enum can implement many interfaces.

### values(), ordinal() and valueOf() methods:

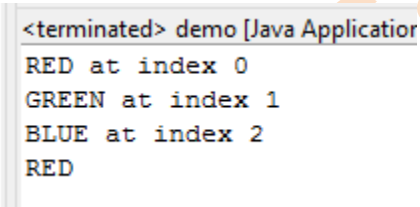
- These methods are present inside **java.lang.Enum**.
- **values()** method can be used to return all values present inside the enum.
- Order is important in enums. By using the **ordinal()** method, each enum constant index can be found, just like an array index.
- **valueOf()** method returns the enum constant of the specified string value if exists.

## CORE JAVA PROGRAMMING

- Java program to demonstrate working of values(), ordinal() and valueOf()

```
enum Color {  
    RED,  
    GREEN,  
    BLUE;  
}  
  
public class demo {  
    public static void main(String[] args)  
    {  
        // Calling values()  
        Color arr[] = Color.values();  
  
        // enum with loop  
        for (Color col : arr) {  
            // Calling ordinal() to find index  
            // of color.  
            System.out.println(col + " at index "  
                               + col.ordinal());  
        }  
  
        // Using valueOf(). Returns an object of  
        // Color with given constant.  
        // Uncommenting second line causes exception  
        // IllegalArgumentException  
        System.out.println(Color.valueOf("RED"));  
        // System.out.println(Color.valueOf("WHITE"));  
    }  
}
```

Output:



```
<terminated> demo [Java Application]  
RED at index 0  
GREEN at index 1  
BLUE at index 2  
RED
```

### enum and constructor:

- enum can contain a constructor and it is executed separately for each enum constant at the time of enum class loading.
- We can't create enum objects explicitly and hence we can't invoke enum constructor directly.

### enum and methods:

enum can contain both **concrete** methods and **abstract** methods. If an enum class has an abstract method, then each instance of the enum class must implement it.

## CORE JAVA PROGRAMMING

- Java program to demonstrate that enums can have constructor and concrete methods.

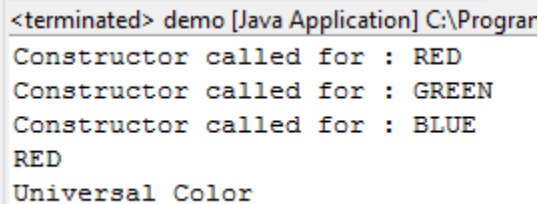
```
//An enum (Note enum keyword inplace of class keyword)
enum Color {
    RED,
    GREEN,
    BLUE;

    // enum constructor called separately for each
    // constant
    private Color()
    {
        System.out.println("Constructor called for : "
            + this.toString());
    }

    public void colorInfo()
    {
        System.out.println("Universal Color");
    }
}

public class demo {
    // Driver method
    public static void main(String[] args)
    {
        Color c1 = Color.RED;
        System.out.println(c1);
        c1.colorInfo();
    }
}
```

Output:



```
<terminated> demo [Java Application] C:\Program
Constructor called for : RED
Constructor called for : GREEN
Constructor called for : BLUE
RED
Universal Color
```

### Difference between enums and Classes

An **enum** can, just like a **class**, have attributes and methods. The only difference is that enum constants are **public, static** and **final** (unchangeable - cannot be overridden).

An **enum** cannot be used to create objects, and it cannot extend other classes (but it can implement interfaces).

### Why and when to Use enums?

## CORE JAVA PROGRAMMING

---

Use enums when you have values that you know aren't going to change, like month days, days, colors, deck of cards, etc.

By - Mr. Gauri Shankar Rai